Web Services Get Tutorial (Java)

In this tutorial, you'll work on a command-line application that displays Tech Elevator locations. The command-line application is partially complete. You'll write the remaining functionality.

Once the application is running, you'll need to call a web API to both get a list of locations and the details for a single application.

Step one: Start the server

Before you start, you need to ensure that the web API is up and running. You need to change directories into the `./server/` folder.

Next, from the command line, run the command `npm install` to install any dependencies. You won't need to do this on any subsequent run.

While still in the command line, run the command `npm start` to start the server. If there aren't any errors, you'll see the following, which means that you've successfully set up your web API:

```
Resources
http://localhost:3000/locations
```

```
 \{^_^}/ hi!

  Loading ./locations.json
  Done

  Resources
  http://localhost:3000/locations

  Home
  http://localhost:3000

  Type s + enter at any time to create a snapshot of the database
  Watching...
```

> You can stop the server, or any other process that you've started from the console, by using the keyboard shortcut `ctrl + c`.

Step Two: Explore the API

Before moving on to the next step, explore the web API using Postman. You can access the following endpoints:

- GET: http://localhost:3000/locations
- GET: http://localhost:3000/locations/{id}

Step Three: Review the starting code

Data Model

There's a class in `/src/main/java/com/techelevator/locations/Location.java` that represents the data model for a location object.

Driver

You'll find the `Main()` method in `/src/main/java/com/techelevator/locations/App.java`.

Provided Code

Also in the `App.java` file, you'll find several private methods that you'll use to print information to the console:

- `printGreeting()`: Prints the welcome greeting along with the menu options
- `printLocations()`: Prints a list of locations
- `printLocation()`: Prints a single location

Your Code

The main method calls a method called `run`. You'll place most of the code you write inside the `run()` method:

```
public static void main(String[] args) {
  run();
}
```

Step Four: Write the console application

In `run()`, you need to create a new `Scanner` instance to read in user input. The `printGreeting()` method displays the greeting and asks the user to select one of the menu options:

```
Scanner scanner = new Scanner(System.in);
printGreeting();
```

If you run the application, you'll see the following:

```
Welcome to Tech Elevator Locations. Please make a selection:
1: List Tech Elevator Locations
2: Exit

Please choose an option:
```

Next, you'll need to read in the user's choice. You can use the scanner to read the next line, which returns a `String`. Then, you need to parse that String into an `int`.

Finally, you'll catch an exception if the user's input can't be parsed to an `int`:

```
  int menuSelection = 0;
  try {
    menuSelection = Integer.parseInt(scanner.nextLine());
  } catch (NumberFormatException exception) {
    System.out.println("Error parsing the input for menu selection.");
  }
  System.out.println("");
```

Now that you have the user's selection, you can decide what action to take next:

```
if (menuSelection == 1) {
  // list locations
} else if (menuSelection == 2) {
  scanner.close();
  System.exit(0);
} else {
  System.out.println("Invalid Selection");
}
```

Step Five: List all locations

If the user selects `1`, you need to list all of the locations returned from the web API. The first thing you'll do is set up a static variable for the `API_URL` because you'll use this several times. Place this above the `main()` method:

```
private static final String API_URL = "http://localhost:3000/locations";
```

Next, you'll create a new instance of the RestTemplate. This is the class that you use to perform a `GET` request to the web API. The locations API returns an array of locations so that's the return type that you are using here.

Finally, call the `printLocations()` method and pass in the array of locations you got back from the API:

```
if (menuSelection == 1) {
  // list locations
  RestTemplate restTemplate = new RestTemplate();
  Location[] locations = restTemplate.getForObject(API_URL,
Location[].class);
  printLocations(locations);
}
```

If you run the application, you should see:

```
Welcome to Tech Elevator Locations. Please make a selection:
1: List Tech Elevator Locations
2: Exit

Please choose an option: 1

-----------------------------------------
Locations
-----------------------------------------
1: Tech Elevator Cleveland
2: Tech Elevator Columbus
3: Tech Elevator Cincinnati
4: Tech Elevator Pittsburgh
5: Tech Elevator Detroit
6: Tech Elevator Philadelphia

Please enter a location id to get the details:
```

Step Six: Get Location Data

In the last step, you returned a list of locations to the user. The last line of the `printLocations()` method asks the user to select a location. When the user selects a location, you'll read in their response:

```
int id = 0;
try {
  id = Integer.parseInt(scanner.nextLine());
} catch (NumberFormatException exception) {
  System.out.println("Error parsing the input for location id.");
}
```

Next, you'll validate the user's input. If the number they enter is greater than 0 and less than the number of locations returned, you can consider it valid. This is because there's a fixed number of results and each result has a corresponding ID. The `if` statement conditional looks like this:

```
if (id > 0 && id <= locations.length) { ... }
```

Now that you have the ID, you can use the `restTemplate` instance that you already created. You'll use this to call the `API_URL` with the ID appended. If you had a chance to test the API in Postman, you know that calling `/locations/1` returns the location data for Tech Elevator Cleveland. Once you have the location, you can pass it to the `printLocation()` method to print it to the console:

```
if (id > 0 && id <= locations.length) {
  Location location = restTemplate.getForObject(API_URL + "/" + id,
Location.class);
  printLocation(location);
```

```
} else {
  System.out.println("Invalid Location Id.");
}
```

If you run the application, you'll see this:

```
Welcome to Tech Elevator Locations. Please make a selection:
1: List Tech Elevator Locations
2: Exit

Please choose an option: 1

-------------------------------------------
Locations
-------------------------------------------
1: Tech Elevator Cleveland
2: Tech Elevator Columbus
3: Tech Elevator Cincinnati
4: Tech Elevator Pittsburgh
5: Tech Elevator Detroit
6: Tech Elevator Philadelphia

Please enter a location id to get the details: 1

-------------------------------------------
Location Details
-------------------------------------------
Id: 1
Name: Tech Elevator Cleveland
Address: 7100 Euclid Ave #140
City: Cleveland
State: OH
Zip: 44103
```

Last but certainly not least, make sure to close the `scanner` and exit the program:

```
scanner.close();
System.exit(0);
```

Summary

In this tutorial you learned:

- How to make an HTTP GET request using Postman and inspect the result
- How to make an HTTP GET request to a RESTful web service using Java process the response
- How to convert a single JSON object into a Java Object
- How to convert an array of JSON objects into an array of Java Objects